



Departamento de Mecatrónica

M3016 Proyecto Integrador de Ingeniería Mecatrónica

M3018 Proyecto Integrador de Ingeniería Mecánica

Control de Robots NAO a través de un navegador web

Profesor asesor:

Dr Luciano Chirinos Gamboa

Profesor supervisor:

Dr Alejandro Rojo Valerio

Alumnos:

Frans Ramirez Neyra	A01362508
Joel Jair López Villalva	A01362464
Isaac Ayala Lozano	A01184862

Semestre

Enero - Mayo 2016

Tabla de contenidos

1. Anteproyecto
 - 1.1. Antecedentes
 - 1.1.1. Planteamiento del problema
 - 1.1.2. Propuesta de solución
 - 1.2. Objetivo General
 - 1.2.1. Objetivos secundarios
 - 1.3. Alcances del proyecto
 - 1.3.1. Entregables
 - 1.4. Metodología
 - 1.4.1. Investigación
 - 1.4.2. Diseño Conceptual
 - 1.4.3. Programación
 - 1.4.4. Documentación
 - 1.4.5. Evaluación
 - 1.5. Calendarización de actividades
2. Marco Teórico
 - 2.1. Robot NAO
 - 2.2. Entorno de desarrollo
 - 2.3. Protocolos de conexión
 - 2.3.1. Conexión TCP
 - 2.3.2. SSH
 - 2.4. PHP
 - 2.5. XAMPP
 - 2.6. WebSockets
3. Desarrollo del proyecto
 - 3.1. Diseño conceptual
 - 3.1.1. Diseño gráfico
 - 3.2. Programación
 - 3.2.1. Página Web
 - 3.2.2. Servidor de Comunicación
 - 3.2.3. Desarrollo del servidor de control
 - 3.2.4. Instalación del servidor
 - 3.3. Funcionamiento del sistema
4. Conclusiones
5. Referencias

Anteproyecto

1.1. Antecedentes

La compañía de Aldebaran Robotics ha contribuido al sector académico con varios robots humanoides. Éstos son empleados frecuentemente en proyectos de programación con un enfoque social. Existen proyectos de investigación que buscan implementar la gama de robots NAO en las área de rehabilitación, y de educación. Sin embargo, son escasos los proyectos que permiten a una persona controlar un robot NAO de manera sencilla.

El área de investigación no padece de problemas al programar estos robots, pues los responsables ya poseen la experiencia necesaria para hacer uso de las herramientas que ofrece la compañía de Aldebaran Robotics. Tampoco tienen complicaciones severas al utilizar las librerías que otros desarrolladores han creado, ya sea para utilizar otros elementos físicos como controles o como sensores, o para agregar funcionalidad a sus programas.

Desafortunadamente, el sector educativo enfrenta dificultades que personas con experiencia y conocimiento raramente encontrarán. Ejemplos de esto son el desconocimiento de cómo hacer uso de librerías de código, cómo comprender la documentación de la plataforma, o qué limitaciones posee el robot. Por ello, para la programación básica de los robots NAO existe Choregraphe.

Choregraphe es una interfaz gráfica de programación para principiantes. Aunque es posible crear rutinas de comportamiento complejas en esta interfaz, el manejo del código no es eficiente y genera problemas de administración. El código se maneja dentro de un único archivo, que crece conforme se extiende el comportamiento programado del robot.

Otro problema que enfrenta el sector educativo es la restricción del tiempo que un alumno puede dedicar a la programación del robot. Como evidencia de esto, se tiene el curso de introducción impartido por Isaac Ayala Lozano a alumnos de preparatoria, del Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Toluca (ITESM). Durante la sesión se dedicó más tiempo a explicar cómo navegar la interfaz gráfica, y a corregir diferencias en la presentación visual de la interfaz que a impartir el curso introductorio.

Son problemas como éstos los que generan la necesidad de crear una plataforma, que permita al sector educativo reducir la cantidad de pasos e instrumentos necesarios para programar los robots. Es decir, es necesario desarrollar una manera más eficiente de controlar al robot NAO según las especificaciones del sector educativo.

Existen ya algunas propuestas que han sido publicadas por universidades alrededor del mundo, como las propuestas de la Universidad Nacional de Seúl (SNU) [1] y de la Université de Fribourg (UNIFR) [2], que intentan resolver estos problemas de diferentes maneras. Ambas

teniendo como punto en común el uso de hardware ajeno a la computadora para realizar las tareas de teleoperación.

La SNU propone el uso de un teléfono celular, con sistema operativo Android como controlador con una aplicación diseñada específicamente para dicho sistema. Su publicación describe una lista de características que satisface las necesidades de los usuarios, los cuales buscan utilizar el robot NAO como un robot teleoperado. La aplicación cuenta con varios indicadores que muestran la situación del robot en tiempo real, así como una retroalimentación visual del robot mediante el uso de las cámaras que el robot posee, los cuáles se muestran en la figura 1.1.

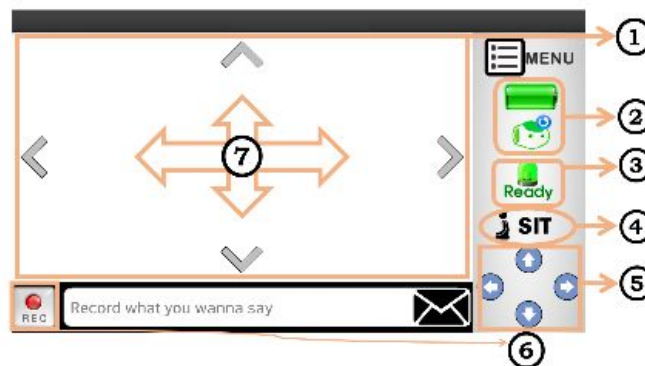


Figura 1.1. Interfaz gráfica de la aplicación de USN.

El otro ejemplo a considerar es la solución obtenida por la UNIFR, la cual hace uso de un conjunto de sensores para lograr controlar de una manera más compleja al robot NAO. Propone el uso de dos Wiimotes, y un micrófono para poder interactuar con el robot de diferentes maneras. Hacen uso de diversas librerías para integrar todas estas tecnologías en una aplicación funcional, que permite al usuario controlar el movimiento y habla del robot. La figura 1.2 muestra el diagrama simplificado de su funcionamiento.

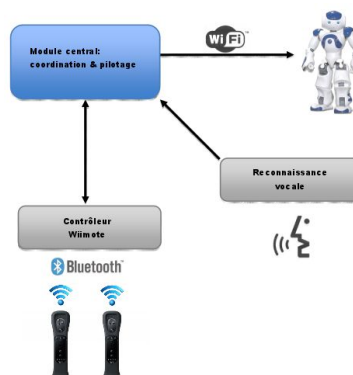


Figura 1.2. Control propuesto por la UNIFR.

Ambas propuestas han demostrado que es posible desarrollar, e implementar soluciones que permiten al usuario utilizar dispositivos que le son más accesibles para controlar al robot NAO.

No obstante, ninguna de las soluciones ofrece la posibilidad de realizar acciones complejas o simultáneas, ni la opción de crear secuencias automatizadas. Es decir, el control se limita a manipular de manera individual los apéndices del robot.

1.1.1. Planteamiento del problema

La programación de robots NAO se ve limitada por la necesidad de tener cerca una computadora con el software necesario; sin embargo, en la práctica esto resulta tedioso ya que es de cumplir con varios requisitos:

- Sistema operativo compatible con el software a utilizar
- Versiones compatibles de programas, sistema operativo y librerías
- Actualizar el sistema del robot NAO para corregir fallas existentes en versiones anteriores

Estos inconvenientes aumentan al desarrollar proyectos en equipo, pues las variables que pueden fallar aumentan a la vez que la probabilidad de que aparezcan aumenta.

1.1.2. Propuesta de solución

Se plantea el uso de una interfaz gráfica accesible mediante un navegador web, en una red local para controlar el robot. Así como ofrecer la posibilidad de programar secuencias de movimiento utilizando movimientos básicos. Con esta propuesta es posible reducir el número de variables que podrían fallar, puesto que la interfaz puede ser usada tanto en celulares como computadoras, con las modificaciones necesarias para adaptarse al tamaño de la pantalla.

1.2. Objetivo General

Crear una interfaz gráfica, accesible en una red local mediante un navegador web compatible, que permita al usuario interactuar y controlar a un robot NAO.

1.2.1. Objetivos secundarios

Como objetivos secundarios se plantean:

- Identificar las acciones de movimiento básicas que se implementarán en la interfaz
- Diseñar la apariencia gráfica de la interfaz web
- Utilizar el API NAOqi para el control del robot NAO
- Diseñar un servicio para administrar la conexión servidor - cliente

1.3. Alcances del proyecto

Los alcances del proyecto se limitan al desarrollo de la interfaz y las pruebas necesarias para confirmar su funcionamiento. A continuación, se presenta un desglose de actividades importantes para la realización del objetivo general.

Interfaz web

- Creación del front-end en HTML5, CSS3 y Javascript
- Creación del back-end basado en GNU/Linux
- Protocolos de conexión entre el front-end y el robot NAO
- Pruebas de conexión entre un navegador web y el front-end
- Pruebas de conexión entre front-end y back-end

Pruebas

- Pruebas de conexión entre una computadora y el servidor
- Pruebas de movimiento comandadas mediante la interfaz

1.3.1. Entregables

Se definen como entregables el código fuente de la interfaz, el back-end y la documentación pertinente para que otras personas interesadas en el proyecto puedan replicar, y modificar las actividades realizadas; así como evidencia de la realización del proyecto y el funcionamiento apropiado de la interfaz.

1.4. Metodología

El proyecto se divide en cinco áreas: Investigación, Diseño Conceptual, Programación, Documentación y Evaluación. Cada área contiene actividades que los diferentes integrantes desarrollarán. La figura 1.3 muestra cómo se realizarán las actividades de cada área.



Figura 1.3. Metodología.

1.4.1. Investigación

El área de investigación comprende todas las actividades relacionadas con recolección de información. Éstas incluyen la búsqueda de ejemplos funcionales, obtención de manuales y archivos de documentación, y la recolección de guías para el diseño gráfico. La investigación engloba tres conceptos importantes: diseño de páginas web, programación del robot NAO, y conexiones entre sistemas.

1.4.2. Diseño Conceptual

El área de diseño conceptual integra las actividades de bosquejado, diseño gráfico, diseño de secuencias de acción y mockups no funcionales. Se hará uso de software para desarrollo de conceptos visuales, diagramas de flujo y recolección de ideas.

1.4.3. Programación

El área de programación se divide en: desarrollo de páginas web, integración de las librerías NAOqi, creación de código complementario, y administración de los proyectos. El desarrollo de páginas web incluye la combinación de lenguajes Python, HTML5, CSS3 y Javascript; con los cuáles se generará todo el sistema que el usuario utilizará para controlar el robot. La integración de librería NAOqi implica el uso de código Python, y la instalación de librerías en los robots NAO para permitir el control del robot.

1.4.4. Documentación

Implica la compilación de todos los avances obtenidos durante el desarrollo del proyecto, así como la generación de la información necesaria para explicar el uso del software. Se busca permitir a otras personas hacer uso del código sin necesidad de tener conocimientos específicos sobre el funcionamiento de la aplicación.

1.4.5. Evaluación

Se engloban aquí todas las actividades relacionadas con presentación de avances, reuniones con el profesor supervisor, reuniones con el profesor asesor, y la presentación final. Implica el desarrollo de reportes, presentaciones y videos para sustentar el avance.

1.5. Calendarización de actividades

La siguiente figura (1.4) muestra la duración de actividades, según su área. El proyecto tiene una duración planeada de tres meses, terminando en la última semana de abril.

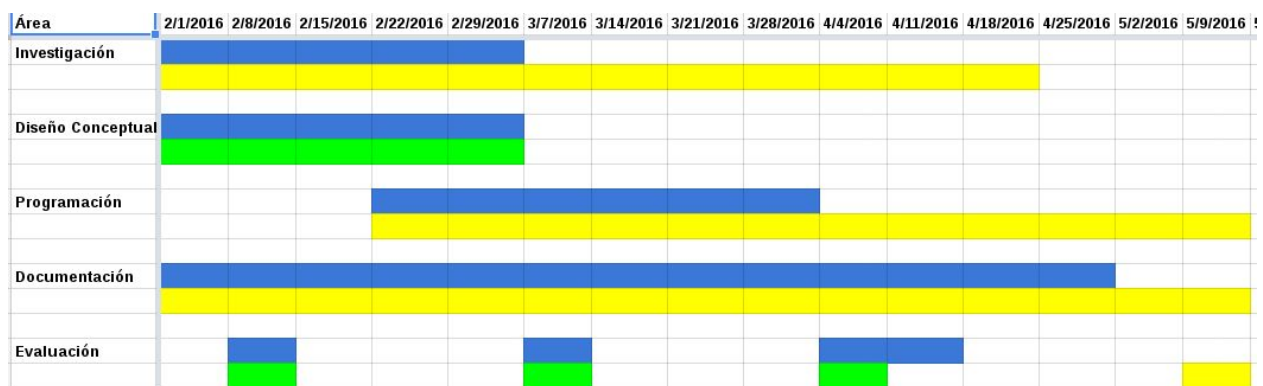


Figura 1.4. Duración del proyecto.

La figura 1.5 presenta las actividades planeadas para el área de Investigación. Involucra la búsqueda de proyectos relacionados con la propuesta, la recolección de código útil para el proyecto y la lectura de documentación.

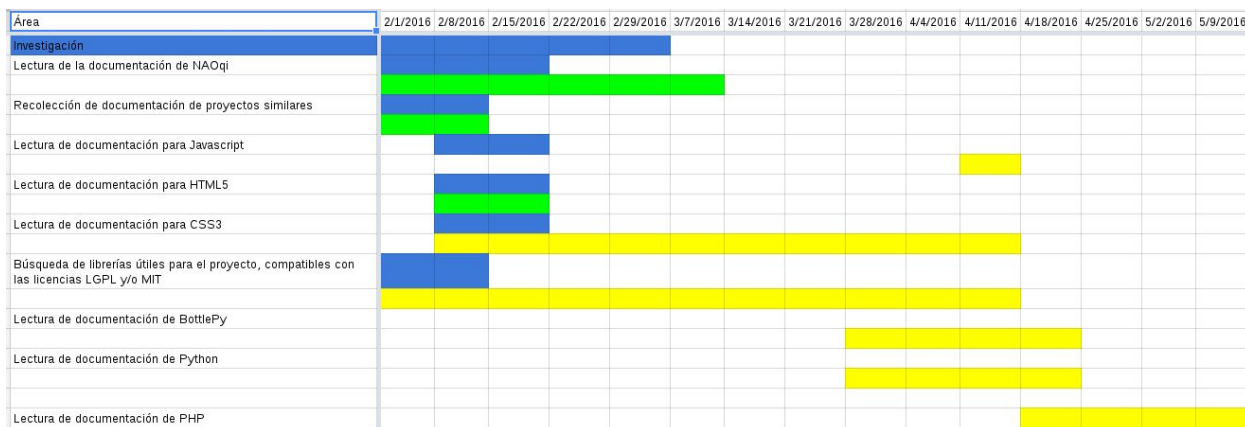


Figura 1.5. Actividades de Investigación.

El área de diseño posee una menor cantidad de actividades que las otras áreas, pero estas actividades son bastante extensas. En la figura 1.6 se observa la calendarización de las tareas propuestas.



Figura 1.6. Actividades de Diseño

Las actividades consideradas para el área de programación se basan en los prototipos que se realizarán en el período de diseño. La interfaz de usuario tomará como referencia el mockup realizado previamente, mientras el back-end se programará en base a los diagramas de flujo que se desarrollen. La figura 1.7 muestra la duración de cada actividad.

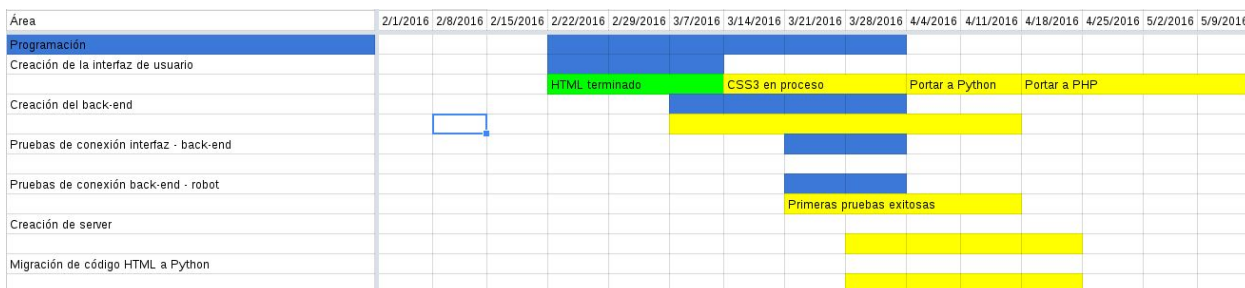


Figura 1.7. Actividades de programación

El área de Documentación se desarrollará durante todo el proyecto, pues sus actividades son semanales y quincenales. La figura 1.8 presenta la duración planeada.



Figura 1.8. Actividades de documentación.

La figura 1.9 contiene las fechas asignadas para el área de evaluación. Se utilizan las fechas asignadas durante el inicio del proyecto para la presentación de cada fase de desarrollo.

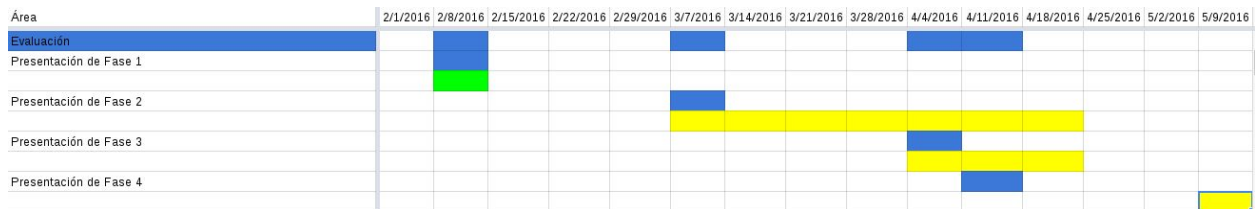


Figura 1.9. Actividades de evaluación.

2. Marco Teórico

2.1. Robot NAO

El robot humanoide usado en este proyecto es el NAO H25 (figura 2.1), diseñado y fabricado por la compañía francesa *Aldebaran Robotics*. NAO es un robot de última tecnología, programable y controlable usando Linux, Windows y Mac; proporcionando una interfaz de comunicación un tanto flexible. Permite realizar movimientos precisos y coordinados. Además, lleva incorporadas una serie de funciones de alto nivel para facilitar su uso.



Figura 2.1 Robot NAO H25

NAO mide 58 cm de altura, pesa 4,3 kg y tiene una carcasa fabricada en plástico. Usa una batería de litio de 21,6 V, con una autonomía aproximada de 45 minutos con el robot en actividad. El consumo en uso normal es de 30W, y en actividad 70W. Permite conexión con cargador de 24V.

Tiene un total de 26 grados de libertad los cuales se muestran en la figura 2.2 y se detallan en la tabla 2.1.

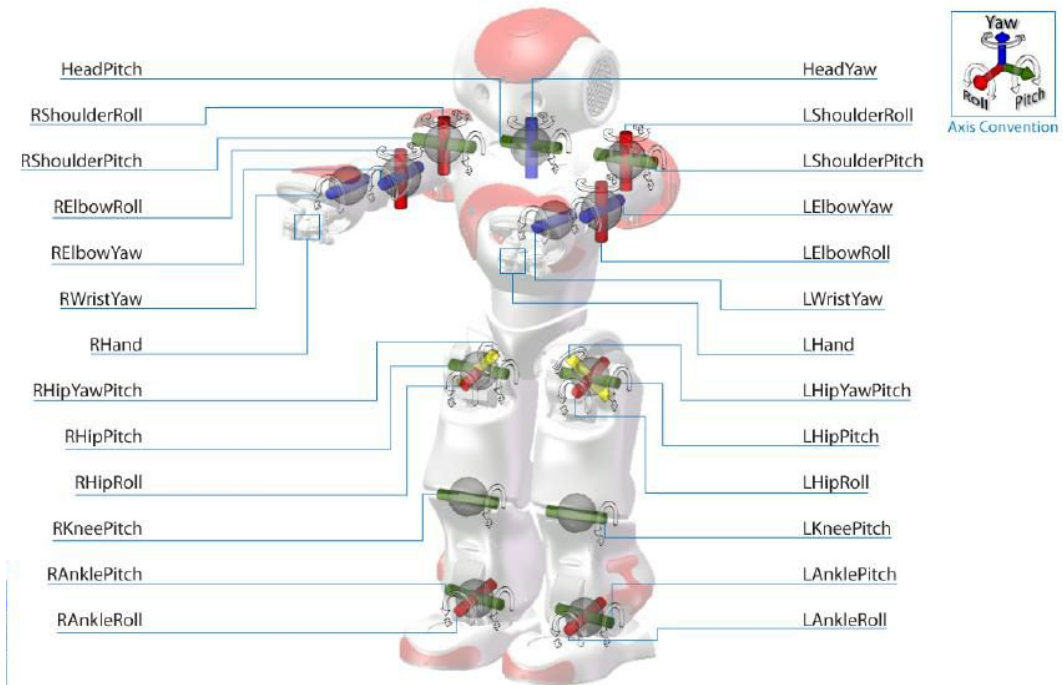


Figura 2.2 Grados de libertad de un robot NAO.

Tabla 2.1. Lista de articulaciones de un robot NAO.

Conector	Articulación	Movimiento	Rango (°)
Head (Cabeza)	HeadYaw (orientación)	Rotación (Z)	-120 a 120
	Headpitch (inclinación)	Flexión/Extensión (Y)	-45 a 45
Larm (Brazo izquierdo)	LShoulderPitch (inclinación del hombro)	Flexión/Extensión (Y)	-120 a 120
	LelbowRoll(giro del codo)	Abducción/aducción (Z)	0 a 95
	LElbowYaw (orientación del codo)	Rotación (X)	-120 a 120
	LelbowRoll (giro del codo)	Flexión/extensión (Z)	-90 a 0
Rarm (Brazo derecho)	RShoulderPitch (inclinación del hombro)	Flexión/Extensión (Y)	-120 a 120
	RelbowRoll(giro del codo)	Abducción/aducción (Z)	0 a 95
	RElbowYaw (orientación del codo)	Rotación (X)	-120 a 120
	RelbowRoll (giro del codo)	Flexión/extensión (Z)	-90 a 0
Lleg (pierna izquierda)	LhipYawPitch (inclinación del hombro)	Giro (Y-Z 45)	-90 a 0
	LhipRoll (giro de cadera)	Abducción/aducción (X)	-25 a 45
	LhipPitch (inclinación de cadera)	Flexión/Extensión (Y)	-100 a 25
	RkneePitch (inclinación de rodilla)	Flexión/Extensión (Y)	0 a 130

		RanklePitch (inclinación de tobillo)	Flexión/Extensión (Y)	-75 a 45
		RankleRoll (giro de tobillo)	Abducción/aducción (X)	-25 a 25
Rleg (pierna derecha)		LhipYawPitch (inclinación del hombro)	Giro (Y-Z 45)	-90 a 0
		LhipRoll (giro de cadera)	Abducción/aducción (X)	-25 a 45
		LhipPitch (inclinación de cadera)	Flexión/Extensión (Y)	-100 a 25
		RkneePitch (inclinación de rodilla)	Flexión/Extensión (Y)	0 a 130
		RanklePitch (inclinación de tobillo)	Flexión/Extensión (Y)	-75 a 45
		RankleRoll (giro de tobillo)	Abducción/aducción (X)	-25 a 25
RHand (mano derecha)		RHand	Abrir/cerrar	
LHand (mano izquierda)		LHand	Abrir/Cerrar	

Referencia:

También incorpora varios sensores y características que se muestran en la figura 2.3, tales como lo son: 2 altavoces de 36mm de diámetro situados en las orejas, 4 micrófonos y 2 cámaras VGA 640x480, 30 fps. Respecto a los sensores, tiene 4 de ultrasonidos (sonar) situados en el pecho, 4 sensores de fuerza en cada pie, sensores de tacto en la parte frontal de cada pie, sensores inerciales (acelerómetro de 3 ejes y giroscopio de 2 ejes), y en cada articulación hay sensores de posición con una resolución de 0.1 °. Dispone de tres sensores táctiles en la cabeza programable y un botón en el centro del pecho también programable. Para hacerlo más vistoso, incorpora LEDs multicolor en los ojos, orejas, pecho y pies. La conexión puede hacerse vía WiFi IEEE 802.11g (inalámbrica) o Ethernet (con cable).

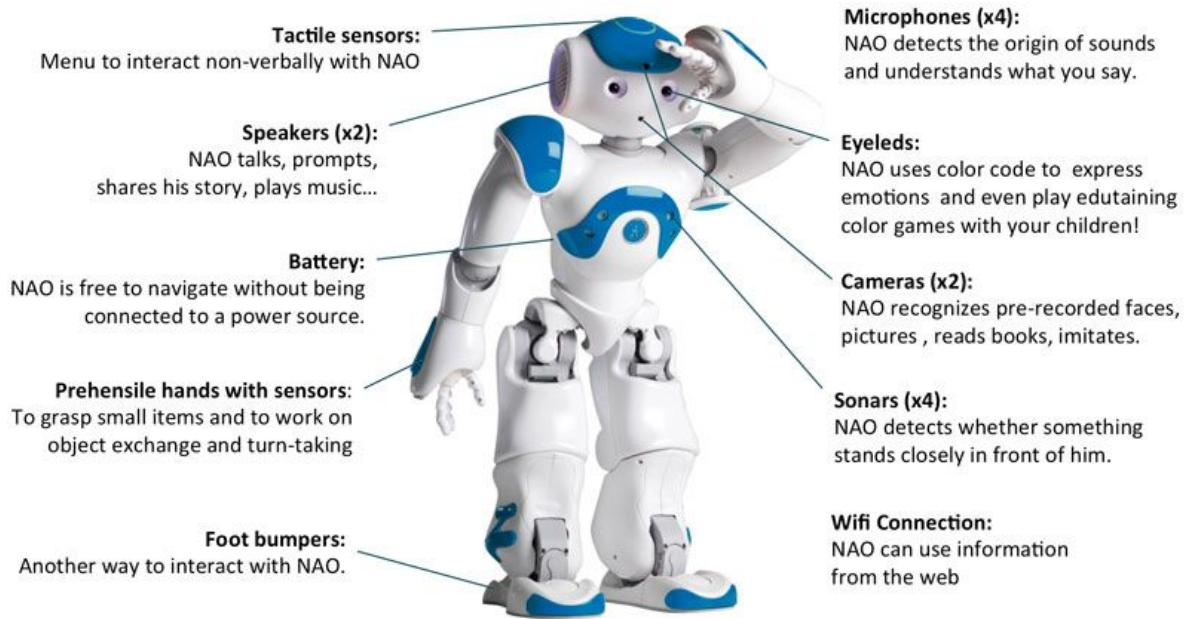


Figura 2.3. Sensores y características de un robot NAO.

Los actuadores son motores *Coreless MAXON DC*. Hay de dos modelos diferentes, y con dos tipos de reducciones diferentes según las necesidades de cada articulación. El control de los motores se realiza mediante microcontroladores PIC que llevan un algoritmo tipo PID (Proporcional, Integral y Derivativo), de parámetros configurables. El modo de interactuar es mediante el envío de instrucciones de posición.

NAO lleva su propia computadora dentro. Tiene un CPU x86 AMD GEODE 500 MHz, con 256 MB SDRAM y 1 GB de memoria flash. El sistema operativo de NAO es un Linux incrustado (32 bit x86), basado en la distribución *OpenEmbedded*. Sobre este sistema operativo corre un programa llamado *NAOQi*, que es el encargado de controlar el robot mediante diversos módulos específicos.

NAO se muestra como un robot con características muy avanzadas. Desde la estructura física, actuadores y sensores hasta la arquitectura de software se nota un diseño muy cuidado. Las funcionalidades que incorpora de control articular, su morfología humanoide bípeda y los múltiples sensores que tiene hacen que sea una plataforma idónea para este tipo de proyectos.

2.2. Entorno de desarrollo

NAO se suministra con un *framework* de desarrollo llamado *NAOQi*. Éste es un programa que implementa una gran cantidad de funciones de alto nivel y se encarga de comunicarse con los dispositivos de bajo nivel. También maneja las comunicaciones. Entre algunas de las funciones de alto nivel que implementa se encuentra un algoritmo de locomoción bípeda.

Para ejecutar órdenes sobre NAO es necesario hacerlo a través de *NAOqi*. Se puede hacer de forma remota (desde una PC) usando lenguajes de programación como C++, Python o URBI. También se pueden compilar programas para ejecutarse desde NAO, programados en C++. Existen también varios entornos de simulación, siendo *Webots* el más usado en este tipo de robots. *Webots* es un entorno de desarrollo para modelar, programar y simular robots móviles. Existen versiones para Windows y Linux. Con *Webots* se puede crear un mundo virtual común con varios robots que interactúen entre ellos y realizar una simulación física precisa gracias al motor matemático ODE (*Open Dynamics Engine*), cabe destacar que esta simulación no es tan precisa cuando se transfieren los resultados al mundo real.

La programación del robot NAO se lleva a cabo utilizando *brokers*, módulos y bibliotecas. La idea fue tener arquitectura distribuida con flexibilidad, modularidad, robustez y que aproveche los recursos computacionales para repartir la carga de la CPU.

NAOqi se ejecuta en el robot por medio de un broker. Un *broker* es un proceso que se encuentra escuchando en una dirección IP y un puerto. Cada *broker* contiene un conjunto de módulos que ofrecen cierta funcionalidad a través de funciones de alto nivel. Cuando se inicia, se carga un archivo de preferencias almacenado en el robot (*autoload.ini*) que define las librerías que se deben cargar. Cada librería contiene uno o más módulos que utilizan el *broker* para acceder a sus métodos, en la figura 2.4 se muestra el proceso de ejecución de NAOqi.

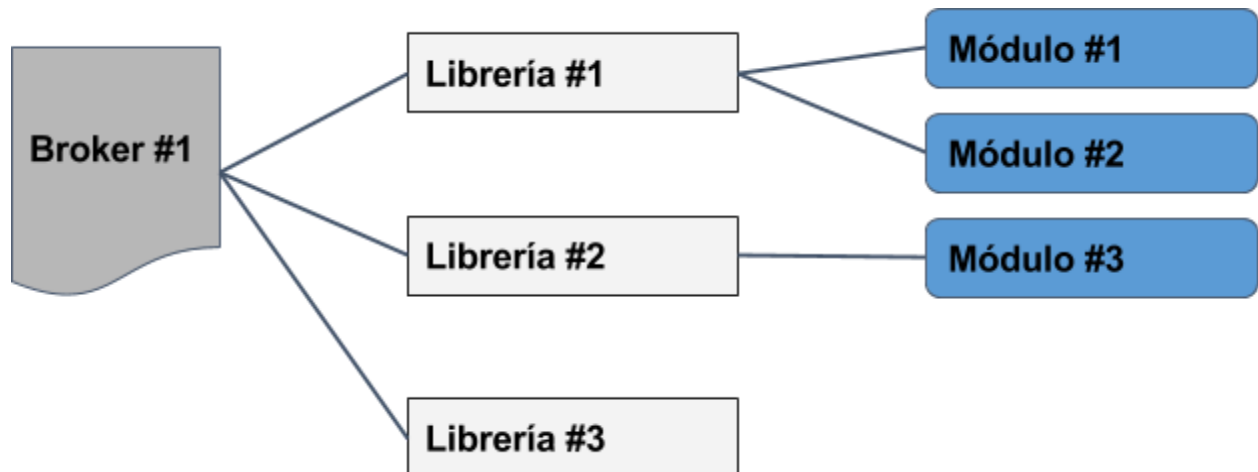


Figura 2.4. Proceso de ejecución de NAOqi.

El *broker* ofrece servicios de búsqueda de modo que cualquier módulo en la red puede encontrar cualquier método que ha sido activado. Los módulos cargados forman un árbol de métodos asociados a los módulos, y módulos conectados a un *broker*, tal como se muestra en la figura 2.5.

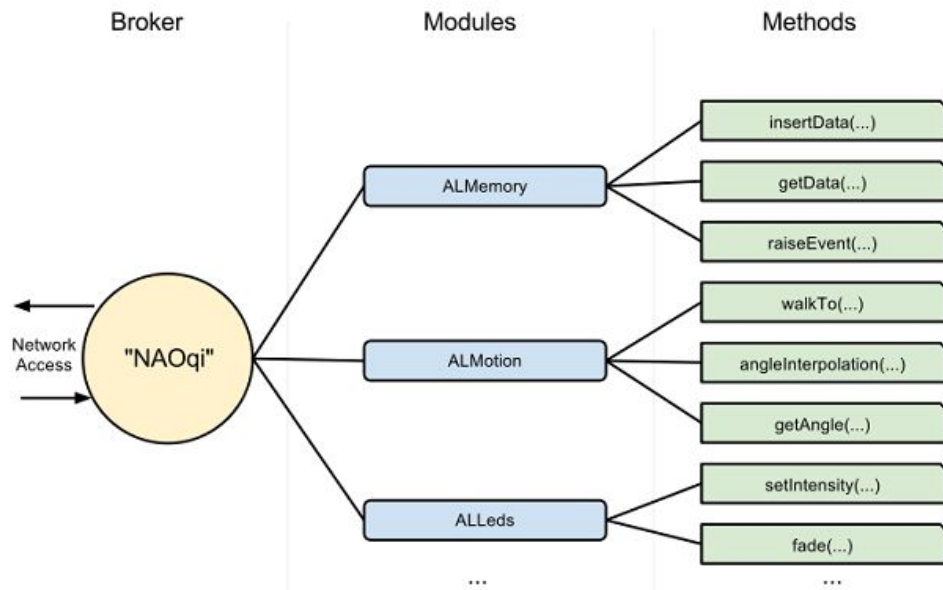


Figura 2.5. Árboles de conexión entre módulos y *brokers* en NAOqi.

El *broker* más importante, y muchas veces el único existente, es el *MainBroker*. Este *broker* contiene módulos que nos permiten acceder a los sensores y actuadores del robot. Cuando se usa el robot real, se lanza un proceso/servicio que se corresponde con el *MainBroker*. En la computadora, ya sea de manera manual o el propio simulador, lanza el proceso *naoqi*, que es realmente el *MainBroker* el cual se encuentra escuchando en el puerto 9559, un ejemplo de la manera en que se ejecutan los brokers se puede visualizar en la Figura 2.6.

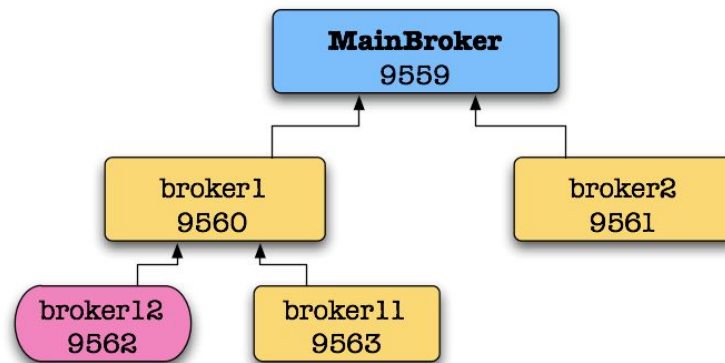


Figura 2.6. Ejemplo de árbol de *brokers* en NAOqi.

Dentro de los módulos del *MainBroker* se pueden encontrar:

- ALMotion, encargado del acceso a los actuadores y a las funciones auxiliares de movimiento.

- ALSensors, encargado del acceso a *bumpers* (sensores de presión situado en la punta de los pies) y el botón del pecho.
- ALSonar, que obtiene información de los sensores
- ALVideoDevice, por el que se puede acceder a la cámara.

Cuando desarrollamos una aplicación podemos decidir si queremos que tenga su propio *broker*, que se conecte al *MainBroker*, o queremos que sea una biblioteca que sea cargada como un módulo propio dentro del *MainBroker*. Cada una de estas maneras tiene sus ventajas y desventajas:

Si tiene su propio *broker*, la comunicación con los módulos del *MainBroker* es más lenta, normalmente esta opción se utiliza para crear un programa ejecutable externo al robot, este utiliza funciones de alto nivel del *Mainbroker* y se conecta a este mediante WiFi. Además se puede reiniciar el *broker* sin necesidad de reiniciar el *MainBroker*. Si el *broker* falla (un fallo de segmentación, por ejemplo), no hará que el *MainBroker* deje de ejecutarse. En este caso tenemos completo control sobre la salida estándar del proceso.

Si se usa como una biblioteca (módulo) del *MainBroker*, la comunicación será más rápida, pero un fallo grave hará que el *MainBroker* termine (y tire el robot al suelo). Para usar esta opción utilizamos una herramienta disponible en *NAOqi* para generar módulos, *module generator*. Los módulos creados se cargarán al robot y se ejecutarán dentro de este.

Cada programa (módulo ó *broker*) actúa como servidor (escucha peticiones a través de un IP y de un puerto) y ofrece servicios apareciendo como módulos, en general cada módulo corresponde a una funcionalidad.

Introducimos ahora el concepto de proxy. Los *proxies* son una herramienta que se utiliza durante la programación. Un *proxy* es un objeto que se comporta como el módulo que representa.

Por ejemplo, si se crea un *proxy* para el módulo *ALMotion* (módulo de movimiento), se obtendrá un objeto que contiene todos los métodos de *ALMotion*.

Para crear un *proxy* para un módulo, (y por lo tanto llamar a los métodos de este módulo) se tienen dos opciones:

- Utilizando el nombre del módulo. En este caso, el código que se desea ejecutar y el módulo al que desea conectarse debe estar en el mismo *broker*. Esto se denomina una llamada local.
- Usando el nombre del módulo, la dirección IP y el puerto de un *broker*. En este caso, el módulo debe estar en el *broker* correspondiente.

Normalmente, cada módulo es una clase dentro de una librería. Cuando la librería se carga desde el *autoload.ini*, automáticamente se crea una instancia del módulo.

Un módulo puede ser local o remoto.

- Si es remoto, se compila como un archivo ejecutable, y se puede ejecutar fuera del robot. Los módulos remotos son más fáciles de usar y se puede depurar con facilidad desde el exterior, pero son menos eficientes en términos de velocidad y en el uso de la memoria.
- Si es local, se compila como una biblioteca, y sólo puede ser utilizado en el robot. Sin embargo, son más eficientes que un módulo de control remoto.

Cada módulo contiene varios métodos. Entre ellos, algunos métodos se pueden llamar desde fuera del módulo, por ejemplo dentro de otro módulo o a partir de un archivo ejecutable. La forma de llamar a estos métodos no varía si el módulo es local o remoto: el módulo se adapta automáticamente.

Naoqi ofrece dos formas de llamar a los métodos:

- **Llamadas bloqueantes:** son las llamadas normales a los métodos, simples llamadas que esperan hasta que el método finaliza. La siguiente instrucción será ejecutada después del final de la llamada anterior. Todas las llamadas pueden generar una excepción y deben ser encapsuladas con un bloque try-catch. Las llamadas pueden tener valores de retorno, véase la figura 2.7.

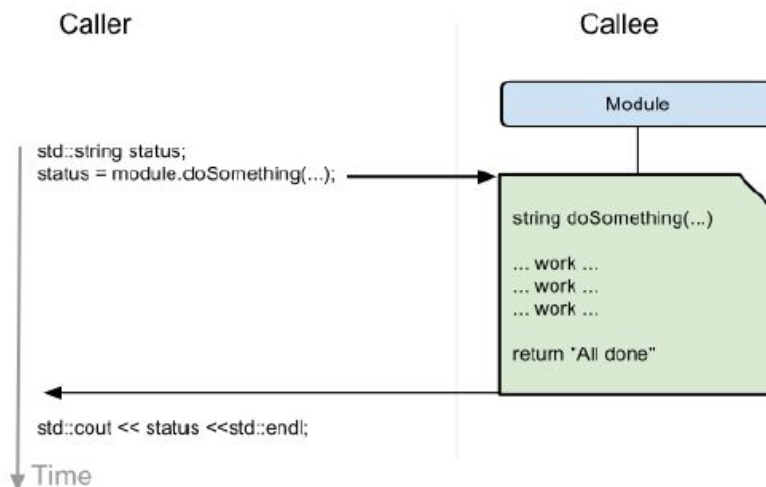


Figura 2.7. Ejemplo de llamada bloqueante en Naoqi.

- **Llamadas no bloqueantes:** se utilizan usando una llamada especial de la que disponen los proxies, la llamada *post*, resaltada en la Figura 2.8. De esta forma crea una tarea en un hilo paralelo. Esto permite realizar tareas paralelas, es decir, al mismo tiempo (por ejemplo, caminar

mientras se habla). Cada llamada posterior genera un identificador de proceso/tarea. Se puede utilizar este identificador de tarea para comprobar si una tarea se está ejecutando, o esperar hasta que la tarea ha terminado.

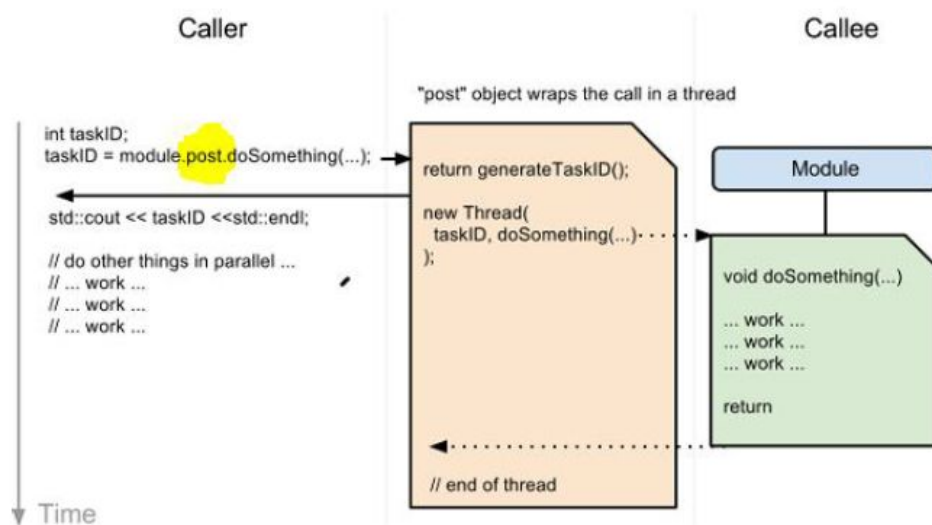


Figura 2.8. Ejemplo de llamada no bloqueante en Naoqi.

El módulo ALMemory es un módulo importante para el entorno de programación NAO ya que obtiene los valores resultantes de sus sensores. Se describirá a continuación sus principales características.

ALMemory es el módulo encargado de gestionar la memoria del robot. Todos los módulos pueden leer o escribir, suscribirse a eventos con el fin de ejecutar cierto código cuando estos suceden.

Hay que tener en cuenta que ALMemory no es una herramienta de sincronización en tiempo real. El límite depende del tipo de suscripción que se haga en el Device Communication Manager (DCM). El DCM es un módulo de NAO, que forma parte del sistema de NAOqi, que se encarga de la comunicación con todos los dispositivos electrónicos en el robot (servos, sensores, actuadores, etc.)

ALMemory es un vector de ALValue (ALValue es la estructura de datos que utiliza NAOqi). El acceso a los datos se realiza de forma segura mediante hilos. Nuestro método utiliza lectura/escritura en secciones críticas para evitar un mal acceso a los mismos (véase Figura 2.9), cuando se lee de la memoria.

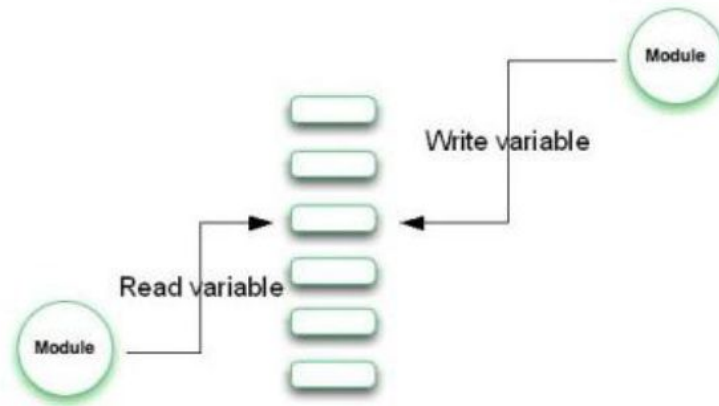


Figura 2.9. Acceso a memoria mediante secciones críticas.

ALMemory contiene tres tipos de datos y proporciona tres APIs diferentes.

- Datos de los sensores y las articulaciones.
- Eventos.
- Micro-eventos.

Datos de los sensores y articulaciones:

- Los datos no tienen historia.
- Son variables de 32 bits.
- Se puede obtener un puntero a la variable con el fin de realizar accesos más rápidos.
- No se puede suscribir a datos.

Unos pocos módulos crean eventos. Para suscribirse al evento de otro módulo, se utiliza una devolución de llamada que debe ser un método de su suscriptor. Por ejemplo, podemos tener un módulo llamado FaceReaction que contiene un método onFaceDetected. Se puede suscribir el módulo FaceReaction al método FaceDetected del módulo ALFaceRecognition con la devolución de llamada onFaceDetected. Esto hará que se active el algoritmo de detección de rostros, y cada vez que se detecta una cara, la devolución de llamada onFaceDetected será llamada.

El módulo de movimiento (ALMotion en NAOqi), controla todas las funciones de alto nivel implementadas para obtener movimientos de las articulaciones del robot NAO. A continuación, se describirán sus principales características.

El módulo ALMotion proporciona métodos que facilitan la creación de movimientos de NAO. Contiene cuatro grupos principales de métodos para controlarlo:

- Métodos que controlan la rigidez (*stiffness*) de las articulaciones, básicamente controlan el encendido y apagado de los motores, además de su fuerza.
- Posición de las articulaciones, interpolación, reactivos de control.
- Métodos para caminar, controlan la distancia, velocidad, posición en el mundo, etc.
- Métodos que controlan los efectores del robot en el espacio cartesiano, cinemática inversa, limitaciones de las articulaciones, etc.

Este módulo ofrece también el acceso a información útil sobre el hardware del robot, tal como el número de articulaciones, su nombre, límites y los sensores disponibles.

La versión actual de NAOqi implementa algunos métodos para evitar la caída del robot, anti-colisión, reflejos a eventos, rigidez inteligente, etc.

Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y en menor medida, programación funcional. Es un lenguaje interpretado y es multiplataforma. Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Para el desarrollo del proyecto se hace uso del lenguaje de programación Python porque es el lenguaje utilizado originalmente en las herramientas y aplicaciones proporcionadas por la empresa Aldebaran, como por el ejemplo Choregraphe.

Si bien es cierto que existe la posibilidad de utilizar otros lenguajes de programación, en el momento de tener que editar algún script proporcionado por la empresa, siempre será más fácil utilizando el lenguaje de programación Python, o como mucho C++, que cualquier otro de los lenguajes con los que tenemos la posibilidad de trabajar al programar los robots NAO. Como valor añadido a la elección tomada, destacar que además, Python es el lenguaje más utilizado por otros desarrolladores de robots NAO, lo que facilitará la incorporación de nuevas funcionalidades futuras desarrolladas incluso por otros desarrolladores si es que se deseara poder utilizar esta opción.

2.3. Protocolos de conexión

2.3.1. Conexión TCP

En el TCP se establecen las conexiones usando el protocolo de acuerdo a tres vías (three-way handshake). Para establecer una conexión, un lado, digamos el servidor, espera pasivamente una conexión entrante ejecutando las primitivas LISTEN y ACCEPT y especificando cierto origen o bien nadie en particular.

En el otro lado, digamos el cliente, ejecuta una primitiva CONNECT especificando la dirección y el puerto IP con el que se desea conectar, el tamaño máximo de segmento TCP que está dispuesto a aceptar y opcionalmente algunos datos de usuario (ejemplo: contraseña). La primitiva CONNECT envía un segmento TCP con el bit SYN encendido y el bit ACK apagado, y espera una respuesta.

Al llegar el segmento al destino, la entidad TCP ahí revisa si hay un proceso que haya ejecutado un LISTEN en el puerto indicado en el campo de puerto de destino. Si no lo hay, envía una contestación con el bit RST encendido para rechazar la conexión.

Si algún proceso está escuchando en el puerto, este proceso recibe el segmento TCP entrante y puede entonces aceptar o rechazar la conexión; si la acepta, se devuelve un segmento de acuse de recibo.

Aunque las conexiones TCP son dúplex integral, para entender la manera en que se liberan las conexiones es mejor visualizarlas como un par de conexiones símplex. Cada conexión símplex se libera independientemente de su igual. Para ello cualquiera de las partes puede enviar un segmento TCP con el bit FIN encendido, lo que significa que no tiene más datos que transmitir. Al reconocerse el FIN, ese sentido se apaga. Sin embargo puede continuar un flujo de datos indefinido en el otro sentido. Cuando ambos sentidos se han apagado se libera la conexión.

2.3.2. SSH

SSH™ (o Secure SHell) es un protocolo que facilita las comunicaciones seguras entre dos sistemas usando una arquitectura cliente/servidor y que permite a los usuarios conectarse a un host remotamente. A diferencia de otros protocolos de comunicación remota tales como FTP o Telnet, SSH encripta la sesión de conexión, haciendo imposible que alguien pueda obtener contraseñas no encriptadas.

Entre sus características tenemos:

- Después de la conexión inicial, el cliente puede verificar que se está conectando al mismo servidor al que se conectó anteriormente.
- El cliente transmite su información de autenticación al servidor usando una encriptación robusta de 128 bits.
- Todos los datos enviados y recibidos durante la sesión se transfieren por medio de encriptación de 128 bits, lo cual los hacen extremadamente difícil de descifrar y leer.

2.4. PHP

PHP es un lenguaje de programación orientado a instrucciones de servidores, lo que lo vuelve la herramienta ideal para controlar las interacciones entre usuarios y robots. Asimismo, el lenguaje permite modularizar toda la interfaz. Con ello se migró el sistema de HTML5 a PHP.

Con la migración se agregaron las instrucciones necesarias, para permitir al servidor de Apache conectarse a los robots NAO. Este código se reutiliza en todas las páginas de comandos, por lo que fue la clave para cumplir el objetivo. Otras funciones del lenguaje son:

- Definición de protocolos de conexión
- Interpretación y traducción de comandos al robot
- Reutilización máxima de código
- Compatibilidad con la configuración del servidor existente

2.5. XAMPP

XAMPP es el acrónimo para un conjunto de software diseñado, para permitir a un programador tener acceso a un servidor local para desarrollo web. El acrónimo significa *Cross-Platform (X)*, *Apache (A)*, *MySQL (M)*, *PHP5 (P)*, *Perl (P)*. Su instalación es bastante sencilla, pues existe un instalador preconfigurado que se encarga del proceso de manera automática.

2.6. WebSockets

WebSockets es una tecnología avanzada que hace posible abrir una sesión de comunicación interactiva entre el navegador del usuario y un servidor. Con esta API, puede enviar mensajes a un servidor y recibir respuestas controladas por eventos sin tener que consultar al servidor para una respuesta.

WebSocket proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor. Debido a que las conexiones TCP comunes sobre puertos diferentes al 80 son habitualmente bloqueadas por los administradores de redes, el uso de esta tecnología proporciona una solución a este tipo de limitaciones proveyendo una funcionalidad similar a la apertura de varias conexiones en distintos puertos, pero multiplexando diferentes servicios WebSocket sobre un único puerto TCP (a costa de una pequeña sobrecarga del protocolo).

3. Desarrollo del proyecto

3.1. Diseño conceptual

3.1.1. Diseño gráfico

El diseño de la interfaz requirió varias iteraciones, cada una debida a distintas razones. Inicialmente se diseñó a mano, y después se realizaron retoques en Inkscape. Se creó el sistema en HTML5 y CSS3, ese modelo se puede observar en la figura 3.1.

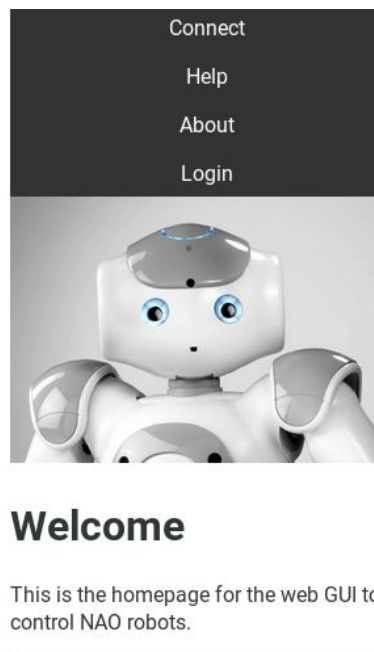


Figura 3.1.1 Diseño inicial.

Después se migró el código a PHP, cambiando el tema y agregando más funcionalidad. Con ello, se finalizó el diseño de la interfaz. La figura 3.2 muestra el resultado.

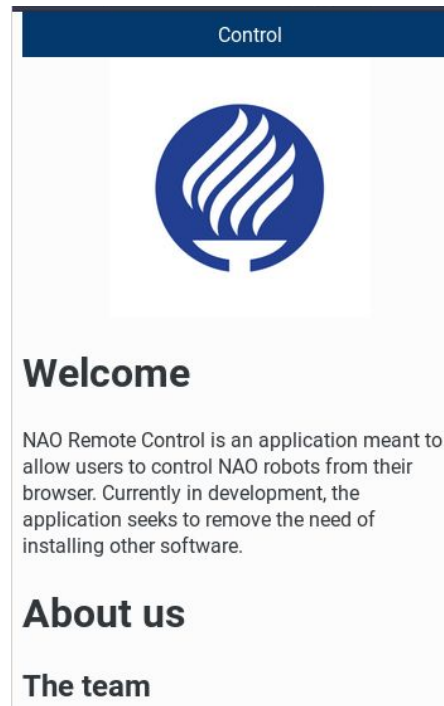


Figura 3.1.2 Diseño final.

3.2. Programación

3.2.1. Página Web

El primer modelo se escribió utilizando exclusivamente HTML5 y CSS3. Se buscó mantener la mayor similitud posible al diseño original. La estructura de la página contenía id's y clases, con el objetivo de separar el diseño gráfico de la funcionalidad del sistema. Fue posible modificar la apariencia de cada elemento desde el código de CSS3, sin causar conflictos con la funcionalidad del programa.

El código se creó con la intención de utilizarlo lo más posible. Por ello, la migración del sistema de un lenguaje a otro no impactó mucho en el proyecto. Al surgir la necesidad de implementar el programa en el servidor, el cambio a PHP fue esencial. Las ventajas de PHP permitieron la reducción de líneas de código de 1346 a 150 líneas aproximadamente. Esta cifra considera solamente la interfaz gráfica, no el resto del sistema.

La razón principal para aprobar la migración del sistema a PHP, fue la funcionalidad de los websockets. Con ello, fue posible establecer una comunicación bidireccional entre el servidor de control, y la página web. Consecuencia de ello, fue que también se pudo adaptar el sistema para la comunicación entre el servidor de conexiones, y el servidor de control.

3.2.2. Servidor de Comunicación

El servidor de comunicación en Nao es un servidor basado en python que se ejecuta en el sistema operativo del Nao, y recibe comandos de la página web.

El servidor almacena diversos módulos, uno para cada comando, y ejecuta ese módulo si recibe el comando especificado en los parámetros. También puede enviar datos al módulo *ALMemory-System* del NAO para iniciar programas individuales de Choregraphe previamente almacenados.

3.2.3. Desarrollo del servidor de control

Para aprovechar la mayoría de las ventajas que ofrece el entorno de desarrollo NAOqi, se ha diseñado un servidor de control escrito en python, el cual está basado en el modelo de Clase & Objeto, Atributo, Servicio, en relación con el modelo MVC es de suma importancia mantener el formato para poder lograr una comunicación efectiva entre las clases y lograr identificar correctamente el dominio de problema, la administración de datos y la interfaz del usuario.

Como base del funcionamiento del servidor, éste se programó usando la tecnología de Python y teniendo como base principal el paradigma orientado a objetos. La clase principal del servidor es *Communication_server.py* la cual se encarga de generar el thread dentro del robot para que se ejecute el servidor. Se cuentan con tres paquetes: el de configuración (settings), el responsable de manejar las conexiones de red (network) y finalmente el responsable de manejar los brokers y módulos de Naoqi para el control de los comandos (commands).

3.2.4. Instalación del servidor

Los archivos de ejecución del servidor se incluyen adjuntos con éste documento, pero iremos desglosando poco a poco el funcionamiento de cada uno de ellos.

Para la instalación del servidor, es necesario copiar todo el contenido de la carpeta *source* en una nueva carpeta llamada *naocom*, dicha carpeta recién creada será necesaria colocarla en la carpeta */home/nao/* dentro del robot, esto se realizó a través de un servidor FTP (FileZilla). Cabe mencionar que los parámetros de conexión son los siguientes y se muestra una captura de la pantalla en la figura 3.2.1:

username: nao

password: nao

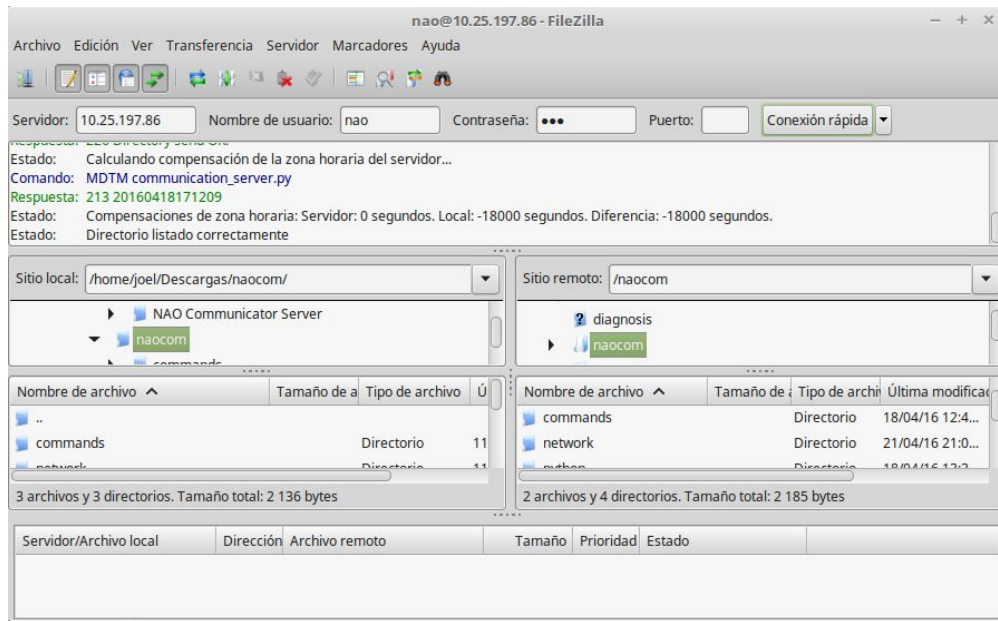


Figura 3.2.1 Captura de pantalla de la aplicación FileZilla.

Una vez lograda la transferencia, será necesario modificar el archivo de arranque del NAO, ésto con el objetivo de que cada vez que el NAO encienda, automáticamente arranque en conjunto el servidor de conexión, ésto lo logramos haciendo un enlace de conexión SSH a través de una aplicación como Putty, tal como se muestra en la figura 3.2.2, ingresando la dirección IP actual del robot con los siguientes parámetros de conexión:

username: nao

password: nao

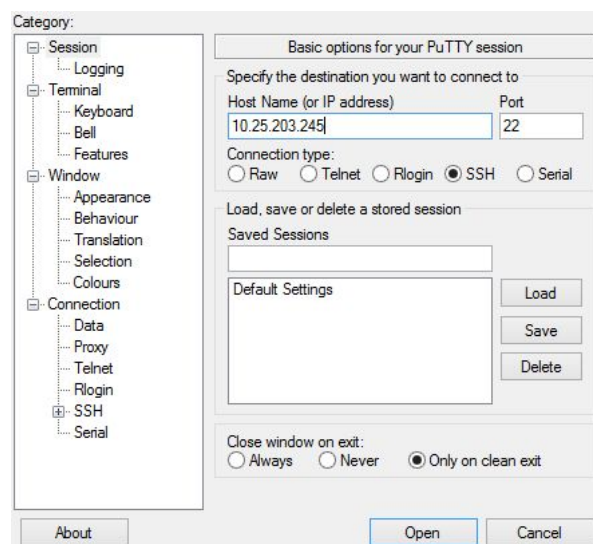


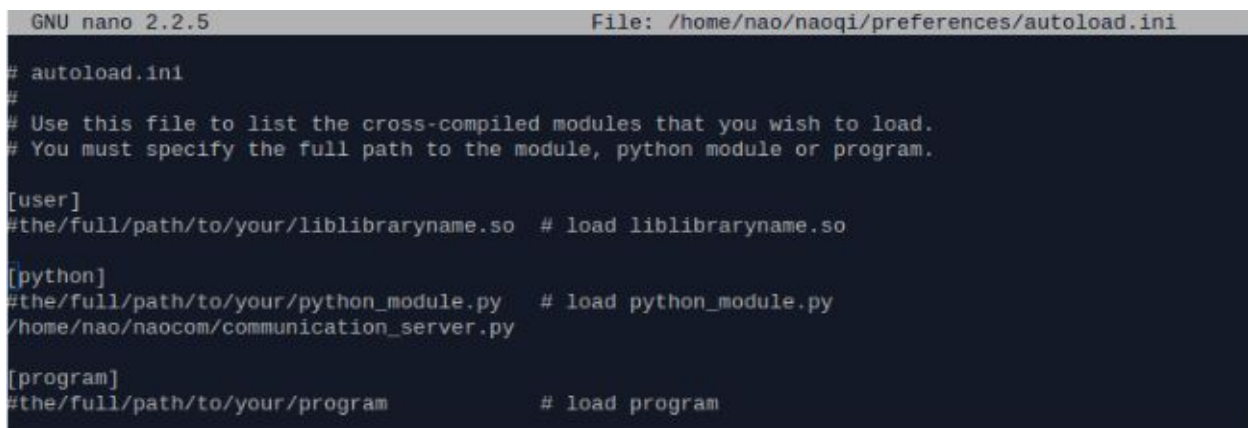
Figura 3.2.2 Captura de pantalla de la aplicación Putty.

Una vez ingresado a la terminal del robot, nos ayudaremos de los siguientes comandos para poder arrancar el servidor en el momento de que NAO encienda.

nano /home/nao/naoqi/preferences/autoload.ini

Se agregó la siguiente línea debajo de la palabra encerrada con corchetes [python]:

/home/nao/naocom/communication_server.py



```
GNU nano 2.2.5 File: /home/nao/naoqi/preferences/autoload.ini
# autoload.ini
#
# Use this file to list the cross-compiled modules that you wish to load.
# You must specify the full path to the module, python module or program.

[user]
#the/full/path/to/your/liblibraryname.so # load liblibraryname.so

[python]
#the/full/path/to/your/python_module.py # load python_module.py
/home/nao/naocom/communication_server.py

[program]
#the/full/path/to/your/program # load program
```

Figura 3.2.3

3.3. Funcionamiento del sistema

El sistema se basa en la secuencia de comunicaciones mostrada en la figura 3.3.1. En ella se observa que el inicio del programa es la página web, en ella están disponibles las acciones que se han implementado en los servidores. El navegador, al seleccionar una acción, manda la instrucción al servidor de conexiones, que la interpreta y traduce para el servidor de control. El comando se envía al robot NAO mediante un websocket y JSON, el cual está siendo monitoreado constantemente por la aplicación instalada en el robot NAO.

El robot NAO, analiza la instrucción recibida y realiza la siguiente secuencia de tareas:

- Provee una respuesta al servidor para confirmar la transmisión de datos
- Evalúa la instrucción
- Ejecuta el comando
- Regresa al modo de escucha

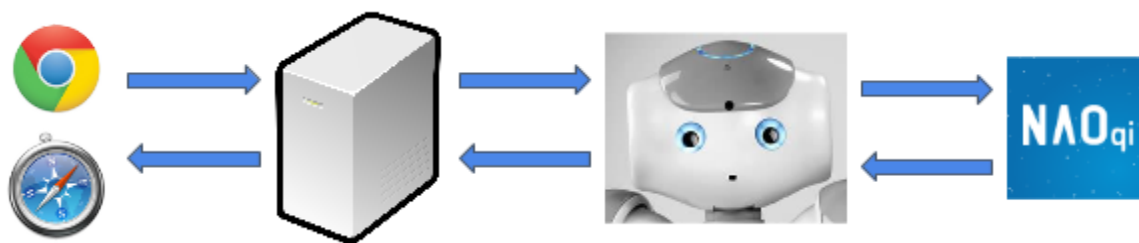


Figura 3.3.1

4. Conclusiones

Se considera cumplido el objetivo del proyecto, puesto que se logró proveer las herramientas necesarias para controlar al robot NAO desde un dispositivo móvil. La plataforma se configuró de manera que otros desarrolladores puedan fácilmente modificar, estudiar y ampliar el catálogo de funciones disponibles en el robot.

El proyecto evolucionó constantemente durante su desarrollo, lo que exigió del equipo un alto grado de dedicación y paciencia. Estos fueron clave para cumplir la necesidad de reescribir grandes cantidades de código, debido a cambios en la configuración del sistema y la distribución física del mismo.

5. Referencias

- [1] Ahn, H., Kim, H., Oh, Y., & Oh, S. (2014). Smartphone-Controlled Telerobotic Systems. Recuperado el 1 de Febrero de 2016, de http://cpslab.snu.ac.kr/publications/papers/2014_CPSNA_SmartNAO.pdf
- [2] Dumoulin, J., Jeanneret, J., Wolf, B., & Wyler, D. (2010). Interfaces multimodales Mini-projet Pilotage NAO. Reecuperado el 28 de Enero de 2016, de https://diuf.unifr.ch/diva/courses/multimodal/projects/nao/mmi_mse_2010_nao.pdf
- [3] Iglesias, Samuel. (2009) Locomoción bípeda del robot humanoide Nao. Recuperado el 3 de Febrero de 2016, de <http://upcommons.upc.edu/bitstream/handle/2099.1/9115/Memoria.pdf>